100 Days of SwiftUI: Final Exam

You scored 100%, which is a pass with distinction – congratulations!

Important: Congratulations on completing the exam! If you want to download your certificate please do so *now* otherwise you will need to re-sit.

Enter your name as it should appear on the certificate

SUBMIT

Click on any answer to show more detail.

1. **Correct**

   Option 1: As it's a container, `VStack` can't have accessibility data.

   Option 2: Three text fields in the same `VStack` are considered to be separate elements if they aren't specifically combined.

   You selected Option 2.

2. **Correct**

   Option 1: The `alignmentGuide()` modifier lets us write custom code to calculate a view's alignment guide.

   Option 2: The presentation mode of a view determines its size on the screen.

   You selected Option 1.

3. **Correct**

   Option 1: We can create a rounded rectangle shape using SwiftUI's `Capsule` shape.

   Option 2: We can place optional views directly into a SwiftUI view hierarchy.

   You selected Option 2.

4. **Correct**

> **Option 1:** SwiftUI lets us animate changes that occur as a result of modifying a Boolean's value.
>
> **Option 2:** Alert messages cannot include string interpolation.
>
> You selected Option 1.

5. **Correct**

> **Option 1:** Properties wrapped in `@EnvironmentObject` must have a value before the view is shown.
>
> **Option 2:** Strings conform to `Identifiable` by default.
>
> You selected Option 1.

6. **Correct**

> **Option 1:** Shapes must be able to create a path.
>
> **Option 2:** SwiftUI coordinator classes must always be nested inside a struct.
>
> You selected Option 1.

7. **Correct**

> **Option 1:** SwiftUI's `Color.red` is not a pure red color.
>
> **Option 2:** A `VStack` can have an alignment or spacing, but not both.
>
> You selected Option 1.

8. **Correct**

> **Option 1:** We can attach `onChanged()` and `onEnded()` modifiers to a `DragGesture`.
>
> **Option 2:** `@EnvironmentObject` only works with structs.
>
> You selected Option 1.

9. **Correct**

> **Option 1:** We can create a `List` directly from an array.
>
> **Option 2:** `@NSManaged` is a property wrapper.
>
> You selected Option 1.

10. **Correct**

> **Option 1:** Apple recommends `@State` properties should use public access control.
>
> **Option 2:** To make a SwiftUI view wrap a UIKit view, we must make it conform to `UIViewControllerRepresentable`.
>
> You selected Option 2.

11. **Correct**

> **Option 1:** Once a `Timer` has started, it can't be stopped.
>
> **Option 2:** Whenever an `@State` property changes, Swift re-invokes our `body` property.
>
> You selected Option 2.

12. **Correct**

> **Option 1:** Only one child view in a given parent can use a custom `layoutPriority()`.
>
> **Option 2:** `EditButton()` will automatically switch between Edit and Done when tapped.
>
> You selected Option 2.

13. **Correct**

> **Option 1:** If we create an `NSManagedObject` subclass, make changes to it, then ask Xcode to create the subclass again, it will add our changes back to the class.

Option 2: The `Codable` protocol is actually a combination of `Encodable` and `Decodable`.

You selected Option 2.

14. **Correct**

Option 1: If a `VStack` has a foreground color and some text inside also has a foreground color, the `VStack`'s foreground color is used.

Option 2: Decorative images are images that are merely there to make the UI look nicer.

You selected Option 2.

15. **Correct**

Option 1: When creating a custom alignment guide we must provide a default value.

Option 2: Arrays cannot be used with `@State`.

You selected Option 1.

16. **Correct**

Option 1: Ease in animations start slow and end fast.

Option 2: SwiftUI's views cannot be created as `@State` properties.

You selected Option 1.

17. **Correct**

Option 1: Images from SF Symbols can be shown larger or small by using the `font()` modifier.

Option 2: Each modifier can be applied only once to a given view.

You selected Option 1.

18. **Correct**

Option 1: If a text field will hold only numbers, we should bind it to an integer property.

Option 2: SwiftUI can animate several properties changing at the same time.

You selected Option 2.

19. **Correct**

Option 1: The `Identifiable` protocol has no requirements.

Option 2: Creating a property using `@Environment(\.horizontalSizeClass)` will keep the value updated when the size class changes.

You selected Option 2.

20. **Correct**

Option 1: The `NavigationBar` view lets us show new views and also place text at the top of the screen.

Option 2: We can send `nil` to the `animation()` modifier.

You selected Option 2.

21. **Correct**

Option 1: QR codes are just barcodes with more colors.

Option 2: The `.fill` content mode might mean parts of an image lie outside its container's frame.

You selected Option 2.

22. **Correct**

Option 1: The `sequenced(before:)` modifier lets us create chains of gestures.

Option 2: `@Binding` cannot be used with a private property.

You selected Option 1.

23. **Correct**

> **Option 1:** SwiftUI allows us to group views into a single accessibility element.
>
> **Option 2:** We can use `@Environment` only once per view.
>
> You selected Option 1.

24. **Correct**

> **Option 1:** We can bind an `alert()` modifier to an optional value.
>
> **Option 2:** SF Symbols don't have any default accessibility labels.
>
> You selected Option 1.

25. **Correct**

> **Option 1:** `@Binding` lets us share one value in two places.
>
> **Option 2:** "Conic gradient" is another name for a radial gradient.
>
> You selected Option 1.

26. **Correct**

> **Option 1:** `ImagePaint` lets us tile an image as a fill or border.
>
> **Option 2:** The `createCoordinate()` modifier lets us create a custom coordinate space.
>
> You selected Option 1.

27. **Correct**

> **Option 1:** A view is always and exactly the same size as its body.
>
> **Option 2:** Child views must always use less than or equal to the amount of space the parent offers them.
>
> You selected Option 1.

28. **Correct**

Option 1: Changing any `@State` property of a view causes SwiftUI to reinvoke the body property.

Option 2: Arrays cannot use the `@Published` property wrapper.

You selected Option 1.

29. **Correct**

Option 1: `contentShape()` allows us to control the tap area for a view.

Option 2: We can show an alert by calling its `show()` method.

You selected Option 1.

30. **Correct**

Option 1: Custom view modifiers must conform to the `ViewModifier` protocol.

Option 2: `ForEach` views can't loop over more than 10 items, because SwiftUI doesn't allow it.

You selected Option 1.

31. **Correct**

Option 1: If we place views inside a `Group` the parent view decides how those views should be laid out.

Option 2: We can have only one `@Published` property in a class.

You selected Option 1.

32. **Correct**

Option 1: SwiftUI stores view positions and sizes as integers.

Option 2: SwiftUI allows no more than 10 child views inside each parent.

You selected Option 2.

33. **Correct**

> **Option 1:** View modifiers must return the same view struct they were given.
>
> **Option 2:** Context menus are triggered when users long press on a view.
>
> You selected Option 2.

34. **Correct**

> **Option 1:** A `GeometryReader` always takes up all available space in its parent.
>
> **Option 2:** Coordinator classes help us respond to actions in a `UIView` or `UIViewController`.
>
> You selected Option 2.

35. **Correct**

> **Option 1:** Before trying to stretch the contents of an image view, we should use `aspectRatio(contentMode: .resize)`.
>
> **Option 2:** We can align text in a `HStack` using the baseline of the first or last text views.
>
> You selected Option 2.

36. **Correct**

> **Option 1:** One environment object can be shared in up to two views.
>
> **Option 2:** When a `URLSession` data task completes, it might send us data or an error, but not both.
>
> You selected Option 2.

37. **Correct**

> **Option 1:** We can use a `ForEach` view inside a `List`.
>
> **Option 2:** When creating a custom `Binding`, we can specify either a `get` closure or a `set` closure, but not both.

You selected Option 1.

38. **Correct**

Option 1: Placing two views in a `List` row will create an implicit `HStack`.

Option 2: Parent views can force a size on one of their chidren.

You selected Option 1.

39. **Correct**

Option 1: Unless we ask for a custom alignment, most parents always place their child views in the top-left corner of their available space.

Option 2: Text views automatically fit the size required to display all their lines.

You selected Option 2.

40. **Correct**

Option 1: We can use `Spacer(minHeight:)` to force a spacer to be at least a certain height.

Option 2: We can force a navigation view to show only one view using `StackNavigationViewStyle`.

You selected Option 2.

41. **Correct**

Option 1: SwiftUI does not let us bind a text field directly to an optional string property.

Option 2: If we want to modify a property, we need to use a SwiftUI property wrapper such as `@Property`.

You selected Option 1.

42. **Correct**

Option 1: We must always return `some View` from a SwiftUI view body.

Option 2: We can call `objectWillChange.send()` to notify SwiftUI that an observable object is about to change.

You selected Option 2.

43. **Correct**

Option 1: We can't use `onDelete(perform:)` with views backed by Core Data objects.

Option 2: We attach code to run when an action sheet button is tapped by providing a closure.

You selected Option 2.

44. **Correct**

Option 1: The `[C]` modifier for `NSPredicate` marks the predicate as being case-sensitive.

Option 2: All SwiftUI views must have a `body` property.

You selected Option 2.

45. **Correct**

Option 1: Accessibility labels must always be a single hard-coded string.

Option 2: The `InsettableShape` protocol builds on the `Shape` protocol.

You selected Option 2.

46. **Correct**

Option 1: iOS can take care of file encryption for using the `.completeFileProtection` option.

Option 2: To read when return is pressed for a text view we should add an `onReturnPressed()` modifier.

You selected Option 1.

47. **Correct**

> **Option 1:** Action sheets can have more buttons than alerts.
>
> **Option 2:** SwiftUI has five built-in coordinate spaces.
>
> You selected Option 1.

48. **Correct**

> **Option 1:** `NavigationView` lets us push a new custom view, or a basic type such as `Text`.
>
> **Option 2:** SwiftUI's lists cannot work with computed properties.
>
> You selected Option 1.

49. **Correct**

> **Option 1:** Buttons must be given a closure to be run when they are tapped.
>
> **Option 2:** Fetch requests must be created using the `@FetchRequest` property wrapper.
>
> You selected Option 1.

50. **Correct**

> **Option 1:** Action sheets can have a title and/or message.
>
> **Option 2:** Swift's `Result` type is designed for use with throwing functions.
>
> You selected Option 1.

51. **Correct**

> **Option 1:** `onDelete(perform:)` cannot be attached directly to a `List` view.
>
> **Option 2:** Timers automatically pause as soon as our app moves to the background.
>
> You selected Option 1.

52. **Correct**

> **Option 1:** We can receive values from a Combine publisher using `onReceive()`.
>
> **Option 2:** We can animate views, but we can't animate view overlays.
>
> You selected Option 1.

53. **Correct**

> **Option 1:** Stacks can have unlimited numbers of views inside them.
>
> **Option 2:** A `GeometryReader` is given one value inside its layout closure, which is a `GeometryProxy` containing layout information.
>
> You selected Option 2.

54. **Correct**

> **Option 1:** Rotating then translating a transform gives the same result as translating then rotating.
>
> **Option 2:** If we want to animate a shape changing, we should add an `animatableData` property.
>
> You selected Option 2.

55. **Correct**

> **Option 1:** The `clipped()` modifier lets us specify a shape for a view should to be drawn inside.
>
> **Option 2:** Swift's `Result` type can contain either success or failure, but not both.
>
> You selected Option 2.

56. **Correct**

> **Option 1:** Asymmetric transitions let us combine transitions with explicit animations.

Option 2: If we want to programmatically set the active tab for a `TabView`, we must set a tag on the views inside it.

You selected Option 2.

57. **Correct**

Option 1: To enable swipe to delete for list rows, we should add an `onSwipeToDelete()` modifier.

Option 2: To remove the label from a date picker, we should use `labelsHidden()`.

You selected Option 2.

58. **Correct**

Option 1: Using `withAnimation()` always uses a spring animation.

Option 2: We can pass data to views inside their initializer.

You selected Option 2.

59. **Correct**

Option 1: One instance of a class can be used in many SwiftUI views.

Option 2: We can't absolutely position views in SwiftUI.

You selected Option 1.

60. **Correct**

Option 1: We can use multiple `animation()` modifiers on a single view.

Option 2: We can apply no more than three modifiers to a single view.

You selected Option 1.

61. **Correct**

Option 1: Classes that are used with `@ObservableObject` must conform to the `ObservedObject` protocol.

Option 2: `AnyView` conforms to `View`.

You selected Option 2.

62. **Correct**

Option 1: If we specify the width of an image, we must also specify its height.

Option 2: One `NavigationView` can show two views inside it.

You selected Option 2.

63. **Correct**

Option 1: Higher layout priority values mean views are more likely to be allocated space in their container.

Option 2: We can make a scroll view take up all available screen width by using `frame(maxWidth: .fill)`.

You selected Option 1.

64. **Correct**

Option 1: Swift has a built-in type for handling dates.

Option 2: It's a good idea to use `drawingGroup()` for all your drawing.

You selected Option 1.

65. **Correct**

Option 1: SwiftUI's buttons require a closure that accepts the button that got tapped as its only parameter.

Option 2: The `Binding` struct is generic.

You selected Option 2.

66. **Correct**

Option 1: SwiftUI ensures `updateUIView()` always gets called at least once a second.

Option 2: Writing data atomically means that iOS writes to a temporary file then performs a rename.

You selected Option 2.

67. **Correct**

Option 1: We can return `nil` from the body of our views.

Option 2: `aspectRatio(contentMode: .fit)` is the same as `scaledToFit()`.

You selected Option 2.

68. **Correct**

Option 1: By default, SwiftUI uses a scale of 1 to 10 for the accessibility values of sliders.

Option 2: When creating a text field we need to provide some placeholder text.

You selected Option 2.

69. **Correct**

Option 1: The order in which we apply modifiers affects the result we get.

Option 2: `sheet()` requires a `NavigationView` to work.

You selected Option 1.

70. **Correct**

Option 1: Gradients must never be used outside the safe area.

Option 2: Text fields have no border by default.

You selected Option 2.

71. **Correct**

Option 1: The `@Binding` property wrapper creates a `Binding` struct.

Option 2: SwiftUI coordinators cannot act as delegates for another class.

You selected Option 1.

72. **Correct**

Option 1: Alerts and action sheets look the same on iPhone.

Option 2: Breaking SwiftUI views into smaller views has little to no performance impact.

You selected Option 2.

73. **Correct**

Option 1: When creating views in a loop, SwiftUI needs to know how to identify each view uniquely.

Option 2: Views presented as sheets automatically share the same environment as the view that presented them.

You selected Option 1.

74. **Correct**

Option 1: The `@Published` property wrapper places our properties inside a `Published` struct.

Option 2: An `@ObservedObject` struct will notify all views that use it when one of its `@Published` properties change.

You selected Option 1.

75. **Correct**

Option 1: We can detect when a sheet is closed by setting its `onClose` parameter.

Option 2: We can embed a `HStack` inside a `VStack`.

You selected Option 2.

76. **Correct**

Option 1: `%@` in an `NSPredicate` is dynamically replaced with a sort order.

Option 2: The `offset()` modifier changes where a view is rendered without actually changing its original dimensions.

You selected Option 2.

77. **Correct**

Option 1: We can let users delete items from a `List` by adding the `onDelete()` modifier to it.

Option 2: If we write `Text("Hello, World!").background(Color.red)`, the text view is a child of the background.

You selected Option 2.

78. **Correct**

Option 1: `@EnvironmentObject` properties must conform to `ObservableObject`.

Option 2: The `disabled()` modifier can read any kind of property, but must not be used with methods.

You selected Option 1.

79. **Correct**

Option 1: By default, a `NavigationView` doesn't work in landscape.

Option 2: We can use the `layoutPriority()` modifier to control how much space a view is allocated.

You selected Option 2.

80. **Correct**

> **Option 1:** Views with an `onTapGesture()` modifier automatically have the `isButton` trait.
>
> **Option 2:** It's possible to mix static and dynamic rows in a `List`.
>
> You selected Option 2.

81. **Correct**

> **Option 1:** All paths are also shapes.
>
> **Option 2:** `GeometryReader` lets us read the size of a view's container.
>
> You selected Option 2.

82. **Correct**

> **Option 1:** We can detect when an `@State` property changes using a property observer.
>
> **Option 2:** `GeometryReader` tells us the size that was proposed by our parent.
>
> You selected Option 2.

83. **Correct**

> **Option 1:** Semantic colors are colors that are named according to their use rather than according to their hue.
>
> **Option 2:** SwiftUI view previews shouldn't have properties of their own.
>
> You selected Option 1.

84. **Correct**

> **Option 1:** `AnimatablePair` lets us animate any two kinds of data.
>
> **Option 2:** `ForEach` views let us loop over ranges and arrays.
>
> You selected Option 2.

85. **Correct**

Option 1: If we use `.accessibilityElement(children: .ignore)` the entire view becomes invisible to VoiceOver.

Option 2: When we import a Core ML model into Xcode, it will automatically generate a Swift class for us to use.

You selected Option 2.

86. **Correct**

Option 1: We can draw borders with a custom shape by using the `overlay()` modifier.

Option 2: We can control the visual appearance of a list using the `listViewStyle()` modifier.

You selected Option 1.

87. **Correct**

Option 1: When the `disabled()` modifier is given a false condition, the view it's attached to stops responding to user interactivity.

Option 2: Colors are views in SwiftUI.

You selected Option 2.

88. **Correct**

Option 1: The `@Published` property wrapper watches an observed object for changes.

Option 2: A coordinator class lets us handle communication back from a UIKit view controller.

You selected Option 2.

89. **Correct**

Option 1: Pickers are always shown as wheels in iOS.

Option 2: We can dynamically replace an `NSPredicate` string with an attribute name using `%K`.

You selected Option 2.

90. **Correct**

Option 1: When `allowsHitTesting()` is false, a view cannot be tapped.

Option 2: SwiftUI disables image interpolation by default.

You selected Option 1.

91. **Correct**

Option 1: Using a value of 100 with the `scaleEffect()` modifier makes a view its natural size.

Option 2: We can attach an `animation()` modifier to a binding.

You selected Option 2.

92. **Correct**

Option 1: If all the properties of a type conform to the `Hashable` protocol, the type itself can also conform just by adding `Hashable` to its list of conformances.

Option 2: When creating a custom alignment guide, it's recommended to use structs rather than enums.

You selected Option 1.

93. **Correct**

Option 1: `UIViewRepresentable` and `UIViewControllerRepresentable` are the same.

Option 2: Segmented controls are created using picker views in SwiftUI.

You selected Option 2.

94. **Correct**

Option 1: In order to be used with `alert(item:)`, a value must conform to `Equatable`.

Option 2: Images built from SF Symbols icons have a customizable foreground color.

You selected Option 2.

95. **Correct**

Option 1: `rotation3DEffect()` can rotate around more than one axis.

Option 2: A view's body can return `View` rather than `some View` in exceptional circumstances.

You selected Option 1.

96. **Correct**

Option 1: Every `VStack` must include one `Spacer` view.

Option 2: Trailing bar button items appear on the right in left-to-right languages.

You selected Option 2.

97. **Correct**

Option 1: We can pop a view from a `NavigationView` using the same presentation mode dismiss code we use for sheets.

Option 2: Using the multiply blend mode usually results in a lighter image.

You selected Option 1.

98. **Correct**

Option 1: The destination of a `NavigationLink` is always shown in the current view.

Option 2: SwiftUI's previews aren't included in our app if we send it to the App Store.

You selected Option 2.

99. **Correct**

Option 1: `NavigationLink` requires a `NavigationView` to work.

Option 2: `Color` is both a view and a shape.

You selected Option 1.

100. **Correct**

Option 1: The `blur()` modifier applies a Gaussian blur to a view, using a radius we specify.

Option 2: We can use implicit animation or explicit animation, but not both.

You selected Option 1.

Total score: 100/100

Back to Review menu

Note: if you're following the 100 Days of Swift or the 100 Days of SwiftUI, just close this window and return to where you were.

@twostraws

paul@hackingwithswift.com